Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Žiga Pirih

3D vizualizacija velikih glasbenih zbirk

diplomsko delo

univerzitetni študijski program
prve stopnje
računalništvo in informatika

Mentor: doc. dr. Matija Marolt

Ljubljana, 2017

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

V okviru diplomskega dela preučite področje vizualizacije glasbenih zbirk. Izdelajte aplikacijo, ki lahko velike zbirke z več milijoni skladb predstavi v 3D prostoru na več nivojih in uporabniku omogoča sprehajanje v tem prostoru. Pri tem naj podobnost med skladbami določa njihovo uvrstitev v prostor, za izvedbo preslikave iz prostora značilk v 3D prostor pa uporabite ustrezno tehniko zmanjševanja dimenzionalnosti podatkov.

Na tem mestu bi se rad zahvalil mentorju doc. dr. Matiju Maroltu za nasvete in potrpežljivost.

Zahvalil bi se tudi vsem ostalim, ki so mi med pisanjem naloge nudili vzpodbudo in nasvete, še posebno svojim staršem.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
GPM	Google Play Music	Glasba Google Play
SNE	Stochastic Neighbor Embedding	Stohastično vgrajevanje sosedov
SOM	Self-organizing maps	Samo-organizirajoči se zemljevidi
MDS	Multi-dimensional scaling	Večdimenzijsko spreminjanje obsega

Povzetek

Naslov: 3D vizualizacija velikih glasbenih zbirk

Avtor: Žiga Pirih

V zadnjih letih je razvoj tehnologije uporabnikom omogočil lažje poslušanje, zbiranje in shranjevanje velikih količin glasbe. Kljub napredku tehnologije pa se sama izkušnja glasbe veliko ni spreminjala, saj večina vmesnikov za predvajanje in upravljanje z glasbo temelji na seznamih.

V okviru diplomskega dela smo izdelali aplikacijo, ki velike glasbene zbirke namesto s seznami predstavi kot pokrajino, zgrajeno na podlagi podobnosti med skladbami. Aplikacija uporabniku omogoča, da s sprehodom po virtualni pokrajini odkriva nove skladbe na podlagi njihove medsebojne podobnosti, med uporabo vmesnika pa se te sproti prenašajo z interneta. Temelji na orodju Unity, v katerem sta implementirana izris pokrajine in uporabniški vmesnik, za prenašanje skladb z interneta pa uporablja pomožne programe, spisane v programskem jeziku Python.

Ključne besede: glasba, vizualizacija.

Abstract

Title: 3D visualisation of large music collections

Author: Žiga Pirih

Technological advances in the last few years allowed users easier access and listening to large amounts of music. But despite the advances in technology, the experience of listening and managing of music has not changed much, as most of the interfaces for this purpose are based on lists.

The purpose of this thesis is developing an application which works with large amounts of music, which are presented using a landscape built on similarities between the songs. The product of the thesis is an application, which allows user to discover similar-sounding songs by exploring the virtual landscape. While user is exploring, the application plays relevant songs, which are downloaded from the internet on the fly. The application is based on the Unity Engine, which renders the environment and provides user interface, while fetching songs from the internet is realized using external programs, which are written in Python.

Keywords: music, visualization.

Poglavje 1

Uvod

V zadnjih nekaj desetletjih je razvoj tehnologije dovolil, da imamo pri sebi vedno več glasbe. Sprva se je glasba iz kaset in plošč preselila na trde diske naših računalnikov ter na naše prenosne naprave. V zadnjih nekaj letih pa so se na trgu pojavili tudi mnogi ponudniki storitev, ki uporabnikom omogočajo, da glasbo poslušajo preko spleta. Izboljševala se je tudi uporabniška izkušnja: z razvojem tehnologije so uporabniki dobili nove možnosti, kot na primer poslušanje skladb iz CD plošč v naključnem vrstnem redu. Ko se je začela glasba seliti v svet digitalnih naprav, so si uporabniki lahko izbrali le tiste skladbe različnih izvajalcev, ki so jim bile všeč ter jih v programu za predvajanje glasbe razvrstili v seznam predvajanja, ki so ga lahko spreminjali glede na svoje razpoloženje. Danes ponudniki pretočnih storitev uporabnikom ponujajo možnost ustvarjanja računalniško generiranih seznamov predvajanja, ki temeljijo na žanru, izvajalcih, ki so uporabniku všeč, podobnostih med skladbami ter izvajalci in drugih značilnostih.

Kljub napredku v tehnologiji pa se sama izkušnja poslušanja glasbe na računalnikih veliko ni spremenila, kajti programi za predvajanje glasbe še vedno večinoma temeljijo na seznamih, na katerih ne prikazujejo kaj več kot naslove pesmi in izvajalca. Nekateri programi ponujajo tudi vizualizacijo trenutne skladbe ter prikaz podatkov, pridobljenih iz interneta, vendar pa podobnosti med skladbami ne vidimo.

Seveda smo bili na področju vizualizacije glasbe že priča nekaterim inovacijam, ki dovolijo uporabniku, da glasbo izkusi na nov način. Eden izmed takih primerov je igra Audiosurf[11], ki na podlagi ene pesmi (po izbiri uporabnika) ustvari pot, ki jo mora uporabnik potem premagati. V sklopu različnih raziskav je bilo tekom let razvitih tudi kar nekaj vmesnikov, ki so v prostoru vizualizirali celotne glasbene knjižnice, vendar pa večina teh ni dostopna javnosti. Poleg nedostopnosti so bili vmesniki, predstavljeni na raziskavah, pogosto preizkušeni le na zbirkah skladb, ki so štele največ nekaj deset tisoč primerkov. To nas je motiviralo, da skušamo narediti aplikacijo z vmesnikom, ki lahko uporabnika popelje po virtualni pokrajini, sestavljeni iz nekaj milijonov pesmi.

Eden izmed razlogov za motivacijo in zanimanje za ustvarjanje pokrajine na podlagi glasbe je bil tudi to, da pri tem gre za neko posebno vrsto proceduralnega ustvarjanja pokrajine. Proceduralno ustvarjanje pokrajin je občasno uporabljeno v igrah — kot na primer Minecraft[16] in Fuel[18] — pa tudi drugod. V teh primerih so pokrajine ustvarjene bodisi z uporabo naključnih števil bodisi matematičnih funkcij bodisi kombinacije obojega.

Naš cilj je ustvariti aplikacijo, ki bo virtualno podlago ustvarila na podlagi podobnosti med milijoni skladb, uporabniku pa dovoljevala prosto raziskovanje pokrajine. Med sprehodom po pokrajini bi uporabnik slišal skladbe, ki so tipične za območje, v katerem se uporabnik nahaja. Zaradi velikega števila skladb je bil eden izmed ciljev tudi ta, da skladb zaradi njihovega števila ne hranimo na disku, temveč jih po potrebi sproti prenašamo z interneta.

V diplomskem delu so sprva predstavljene obstoječe rešitve ter teoretična podlaga našega dela, zatem pa opis poglavitnih tehnologij, ki smo jih uporabili pri razvoju aplikacije. V 4. poglavju sledi temeljit opis strukture naše aplikacije.

Poglavje 2

Teoretična podlaga in obstoječe rešitve

2.1 Pregled že obstoječih rešitev

V sklopu tega diplomskega dela smo se osredotočili predvsem na to, kako različne pesmi umestiti v prostor. Zaradi tega se nismo posvečali različnim načinom, kako iz skladb izluščiti podrobnosti oziroma značilke. Bolj podrobno pa smo pogledali, kako iz danih značilk izračunamo podobnosti med skladbami ter jih umestimo v prostor. Do sedaj so se v podobnih raziskavah uporabljali predvsem trije pristopi: [15]

Samoorganizirajoče karte[13] (angl. self-organizing maps, SOM) Ta pristop uvršča podatke v vnaprej določene gruče. Poglavitna prednost tega pristopa je enostavno vključevanje novih podatkov v obstoječ model, a je nelinearen in slabše ohranja razdalje med predmeti. [17]. Ta pristop je uporabljal vmesnik nepTune[5], ki je bil navdih za naše delo. Vmesnik je prikazan na sliki 2.1.

Večdimenzijsko spreminjanje obsega (angl. multidimensional scaling, MDS) Pri tem pristopu se podatki prosto porazdelijo po prostoru. Ker podatki niso omejeni v vnaprej določene gruče, ta pristop bolje ohranja

Slika 2.1: Vmesnik nepTune[6].

razdalje med podatki. Vmesniki, ki so uporabljali ta pristop, so prinesli rezultate, ki so bili videti bolj naravno kot pri uporabi SOM. Za ta pristop obstaja več različnih izvedenk. Primer vmesnika, ki se poslužuje tega vmesnika, je MusicGalaxy[21].

t-SNE Tehnika t-SNE[25] je različica stohastičnega vstavljanja sosedov (angl. Stohastic Neighbor Embedding, SNE). Podobno kot MDS, t-SNE umešča podatke v prostor na način, ki ohranja podobnosti med različnimi kosi podatkov, vendar pa je tehnika t-SNE pri preslikovanju v nižje dimenzije zmožna bolje ohranjati tako lokalno kot globalno strukturo podatkov. Ta pristop je uporabljal vmesnik, opisan v [23].

Po načinu vizualizacije so se vmesniki delili na dva dela:

Vizualizacija v 2D prostoru – ti vmesniki so bili preprosti, skladbe pa so prikazovali v dveh dimenzijah. Skladbe so bile prikazane kot točke na ravnini.

Vizualizacija v 3D prostoru – nekateri vmesniki — kot na primer nepTune — so glasbeno zbirko predstavili kot pokrajino v 3D prostoru. Vmesnik ni prikazoval posameznih skladb, temveč le mesta, kjer so se pojavljale gruče skladb.

Opazili smo tudi, da je bilo za večino vmesnikov rečeno, da so zmožni upravljati z velikimi zbirkami podatkov, vendar pa se to ni vselej odražalo v demonstrativnih primerih. Razpon števila skladb v njih je segal od nekaj sto do nekaj deset tisoč skladb.

2.2 Značilke skladb

Zvok ima več komponent (npr. ritmična, harmonična, barvna itd.), za njih pa lahko izračunamo vrednosti, ki jih opisujejo. Tem vrednostim pravimo značilke. Naše delo smo osnovali na bločnih značilkah[19], ki se računajo po posameznih blokih.

Pri računanju bločnih značilk se zvočni spekter najprej pretvori v logaritemsko lestvico in normalizira. Nato se razdeli na bloke. Vsak blok vsebuje neko (stalno) število vzorcev, ki definirajo širino bloka. Vsak naslednji blok je od začetka prejšnjega bloka odmaknjen za določen odmik: če je ta odmik dovolj majhen, se lahko bloki med seboj prekrivajo; če pa je odmik dovolj velik, so lahko med bloki tudi vzorci, ki jih zavržemo.

Dobljene bloke na koncu posplošimo v vrednosti, ki predstavljajo celotno skladbo. To storimo tako, da si iz vseh blokov za vsako dimenzijo značilke izberemo vrednost na določenem kvantilu (npr. na mediani).

Za vsako pesem smo imeli naračunanih več različnih vrst značilk[20]:

Spektralni vzorec (angl. spectral pattern) — za to značilko so zajeti kratki bloki frekvenčnega spektra.

Delta-spektralni vzorec (angl. delta spectral pattern) — pri računanju te značilke vzamemo razliko med dvema kosoma frekvenčnega spektra, ki sta zamaknjena za nekaj vzorcev.

Variančni delta-spektralni vzorec (angl. variance delta spectral pattern) — tudi pri tej značilko vzamemo dva kosa frekvenčnega spektra, ki sta zamaknjena za nekaj kosov. Od delta-spektralnega vzorca se razlikuje le po tem, da namesto razlike gledamo varianco.

Vzorec logaritmičnega nihanja (angl. logarithmic fluctuation pattern) — predstavlja ritmično strukturo pesmi.

Korelacijski vzorec (angl. correlation pattern) — prikazuje korelacijo med različnimi frekvenčnimi pasovi. Izračunamo ga tako, da primerjamo vsak par frekvenčnih pasov in zanj izračunamo korelacijski koeficient. Rezultat je simetrična korelacijska matrika.

Vzorec spektralnega kontrasta (angl. Spectral contrast pattern) — Ta značilka prikazuje "*tonskost*" spektralnega okvirja. To je izvedeno tako, da se v vsakem spektralnem okvirju izračuna razlike med spektralnimi vrhovi in dolinami v več pasovih. Ker se spektralni vrhovi v grobem ujemajo s tonalnimi komponentami, sploščena območja v spektru pa so pogosto povezana elementi, ki so podobna šumu, nam razlika med vrhovi in dolinami pove tonskost skladbe.

Pri računanju značilk smo uporabili le spektralni vzorec, ostalih značilk pa nismo uporabili, saj bi bilo računanje podobnosti pri uporabi vseh značilk preveč računsko in prostorsko zahtevno.

2.3 t-SNE

Tehnika t-SNE[25] je različica stohastičnega vstavljanja sosedov (angl. Stohastic Neighbor Embedding — SNE), ki vsakemu vhodnemu podatku dodeli položaj v dvo- ali tridimenzionalnem prostoru (zemljevid). Od ostalih izpeljank SNE jo je precej lažje optimizirati. Ker skuša zmanjševati zbiranje točk na sredini zemljevida, so vizualizacije, pridobljene z uporabo te metode, precej boljše kot pri uporabi alternativnih metod. Prav tako je t-SNE boljša od drugih obstoječih metod, ko želimo v prostoru prikazati strukture na več različnih nivojih. To je še posebno zaželeno, ko želimo v nižjedimenzijski prostor postaviti podatke višjih dimenzij, ki ležijo na različnih, a povezanih mnogoterostih.

Običajna SNE deluje tako, da pretvarja evklidske razdalje med vhodnimi podatkovnimi točkami v pogojne verjetnosti, ki predstavljajo podobnosti. Podobnost med dvema podatkovnima točkama x_i in x_j je pogojna verjetnost $p_{j|i}$, da bi x_i za soseda izbrala x_j, če bi bili sosedje izbrani v sorazmerju z njihovo verjetnostno gostoto pod Gaussovo porazdelitvijo s sredino na točki x_i. Za bližnje točke je ta verjetnost visoka, za zelo oddaljene zelo majhna.

Matematično je pogojna verjetnost $p_{j|i}$ definirana kot:

$$p_{j|i} = \frac{exp(- \parallel x_i - x_j \parallel^2 /2\sigma_i^2)}{\sum_{k \neq i} exp(- \parallel x_i - x_k \parallel^2 /2\sigma_i^2)}$$

kjer je σ_i definirana kot varianca Gaussove porazdelitve, usredinjene na točki x_i. Za točki y_i in y_j, ki v nižjih dimenzijah pripadata podatkovnima točkama x_i in x_j, je podobna pogojna verjetnost $q_{j|i}$ definirana kot

$$q_{j|i} = \frac{exp(- \parallel y_i - y_j \parallel^2)}{\sum_{k \neq i} exp(- \parallel y_i - y_k \parallel^2)}$$

SNE minimizira vsoto Kullback-Leiblerjevih divergenc preko vseh podatkovnih točk z metodo gradientnega sestopanja. Cenilna funkcija (angl. cost function) je podana kot

$$C = \sum_i KL(P_i \parallel Q_i) = \sum_i \sum_j p_{j|i} log \frac{p_{j|i}}{q_{j|i}}$$

P_i tu predstavlja razporeditev pogojne verjetnosti preko vseh podatkovnih točk glede na točko x_i, Q_i pa predstavlja razporeditev pogojnih verjetnosti preko vseh točk na zemljevidu glede na točko y_i.

Gradient, ki ga dobimo z minimizacijo cenilne funkcije, si lahko predstavljamo kot vzmeti med točko y_i in ostalimi točkami y_j, ki potiskajo v smeri $(y_i - y_j)$. Vzmet odbija oziroma privlači dve točki v prostoru glede na to, ali sta preblizu oziroma predaleč, da bi predstavljale podobnosti med dvema točkama v višjih dimenzijah.

t-SNE ta pristop spremeni tako, da za računanje podobnosti med dvema točkama v nizko-dimenzijskem prostoru namesto Gaussove porazdelitve uporablja Studentovo t-porazdelitev. Poleg tega pa uporablja simetrično različico

SNE, ki minimizira Kullback-Leiblerjeve divergence, ne med pogojnimi verjetnostni med $p_{j|i}$ in $q_{j|i}$, temveč med skupno verjetnostno porazdelitvijo. Ker je gradient v simetrični SNE preprostejši, je računanje hitrejše, v praksi pa nima negativnega vpliva na rezultate.

2.4 Metode za interpoliranje

Eden izmed bolj preprostih načinov za ustvarjanje naključnih pokrajin in slik je ta, da se na podlagi matematičnih funkcij ali naključnega šuma ustvari več slik oziroma višinskih zemljevidov v različnih velikostih. Ko so slike oziroma višinski zemljevidi ustvarjeni, jih povečamo na enako velikost ter na koncu združimo v eno samo sliko oziroma višinski zemljevid. Ko ustvarjamo sliko oziroma teren po tem postopku, si moramo izbrati metodo za interpolacijo, z uporabo katere bomo spremenili velikost slik. Teh metod je veliko, spodaj pa so naštete nekatere izmed najbolj pogosto uporabljenih[14][22]:

Najbližji sosed Pri tej metodi za vsak piksel v povečani (oziroma pomanjšani) sliki izračunamo njegovo mesto na izvorni sliki. Za vrednost novega piksla uporabimo vrednost tistega piksla na izvorni sliki, ki je najbližji izračunani točki. Slabost te metode je, da je slika, kateri smo velikost spremenili z uporabo te metode, popačena in kvadratasta; če sliko zmanjšujemo, pa lahko pride tudi do pojava prekrivanja (aliasing). Izjema je povečevanje slike za celoštevilski faktor, ko ta metoda spremeni velikost pikslov brez izgube kakovosti slike (povečano sliko lahko z uporabo te metode zopet pomanjšamo na izvorno velikost. Rezultat pomanjšanja bo enak izvorni sliki). Pri uporabi celoštevilskega faktorja povečave je slabost te metode hkrati tudi njena prednost, saj je v določenih primerih kvadratast videz slike zaželen (oziroma vsaj subjektivno boljši od bolj zabrisanega videza, ki ga dajejo druge metode). Poleg tega je ta metoda najhitrejša in najpreprostejša za implementacijo.

Bilinearna interpolacija – pri tej metodi za vsak piksel v novi sliki izračunamo njegovo mesto na izvorni sliki ter vzamemo vrednosti tistih štirih pikslov, ki ležijo najbližje izračunani točke, pri čemer je vpliv posameznega piksla na končno vrednost točke obratno sorazmeren z njegovo oddaljenostjo: bližje, kot je piksel točki, večji vpliv ima na njeno vrednost.

Bilinearna interpolacija se izvede tako, da si izberemo para točk, ki ležita vzdolž ene izmed osi, ter se med njima na mestu, vzporednem s točko, izvede interpolacijo. Rezultat sta dve začasni vrednosti, med katerima izvedemo linearno interpolacijo še vzdolž druge osi.

Povečana slika ima zabrisane, a še vedno rahlo nazobčane robove. Meje med piksli so opazne. Daje malo boljše rezultate kot metoda najbližjega soseda. Je malo počasnejša, vendar še vedno dovolj hitra.

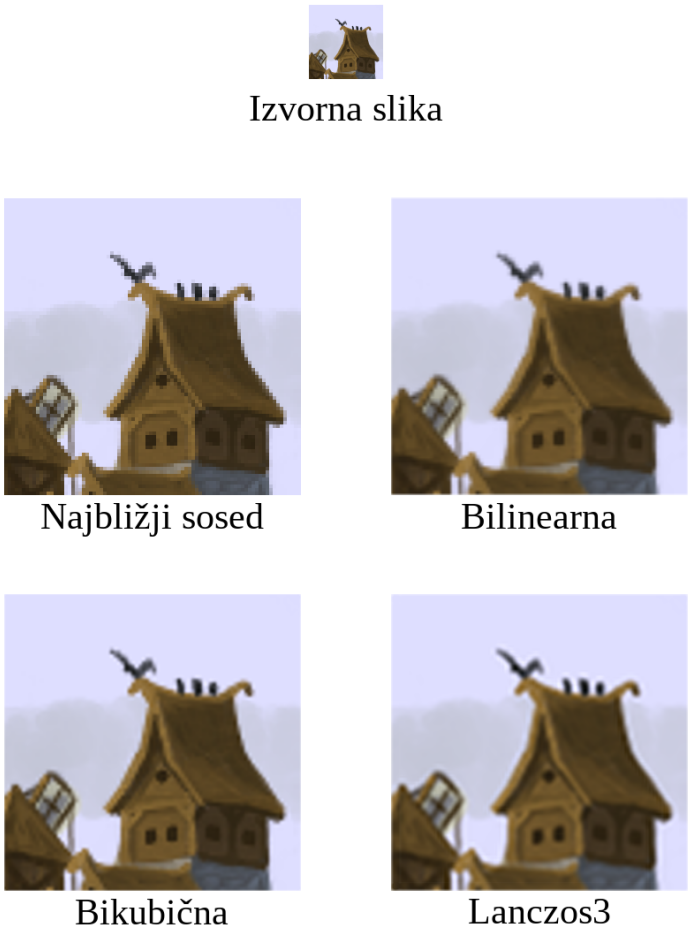
Bikubična interpolacija Ta metoda je podobna bilinearni interpolaciji, le da tokrat namesto najbližjih štirih točk izvorne slike vzamemo najbližjih šestnajst. Vrednost novega piksla je izračunana glede na razdaljo teh pikslov od izbrane točke, pri čemer imajo bližji piksli večji vpliv na vrednost novega piksla. Rezultat interpolacije je zvezna krivulja, zaradi česar je povečana slika videti ostrejša kot pri uporabi bilinearne interpolacije, robovi pa so manj nazobčani. Ta metoda je počasnejša od prejšnjih dveh.

Lanczosovo vzorčenje/sinc Za interpolacijo pri tej metodi uporabljamo Lanczosovo jedro, ki temelji na matematični funkciji sinc. Jedro lahko opišemo s funkcijo:

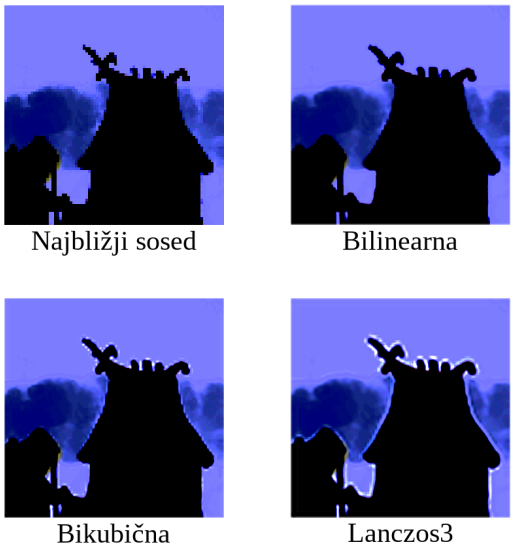
$$L(n) = \begin{cases} sinc(x)sinc(x/a) & , |x| < a \\ 0 & , \text{sicer} \end{cases}$$

Parameter a definira velikost okna, njegova vrednost pa je tipično 2 ali 3. Interpolacijo izračunamo tako, da za vsako točko izračunamo vsoto

$s_i L(x-i)$ na intervalu od $\lfloor x \rfloor - a + 1$ do $\lfloor x \rfloor + a$. V dveh dimenzijah vrednost Lanczosovega jedra izračunamo po formuli $L(x, y) = L(x)L(y)$. Prednosti povečevanja z uporabo Lanczosovega vzorčenja so bolj gladki in nenazobčani robovi, medtem ko slika ni preveč zabrisana. Slabosti tega pristopa so — poleg tega, da je od vseh tu naštetih najbolj počasen — različni artefakti, kot na primer prevelike oziroma premajhne vrednosti tik ob mestih, kjer se barvne vrednosti hitro spreminjajo. Primer takega artefakta je viden tudi na sliki 2.2, kjer se na levi strani hiše v spodnjem desnem primeru na oblaku tik ob robu strehe pojavi tanka, svetla črta; na sliki 2.3 je ta pojav še dodatno poudarjen.



Slika 2.2: Primer različnih načinov interpolacije. Vzeli smo kos slike in ga v programu GIMP z uporabo različnih načinov interpolacije povečali za 4-krat.



Slika 2.3: Če povečamo kontrast slike 2.2 artefakti, ki nastanejo z uporabo Lanczosovega vzorčenja, postanejo še bolj očitni. Podobne artefakte je moč opaziti tudi pri bikubični interpolaciji, vendar pa v veliko manjši meri.

Poglavje 3

Uporabljene tehnologije, orodja in viri podatkov

3.1 Tehnologije in orodja

3.1.1 Unity

Unity[8] je eno izmed vodilnih orodij za izdelavo računalniških iger. Za študente in osebno uporabo je zastonj, poleg tega pa je tudi enostavno za uporabo. Programiranje aplikacij z Unity je možno v programskem jeziku C#, javascriptu in jeziku UnityScript. Unity nam zagotavlja stroj za upodabljanje ter nekaj osnovnih gradnikov, ki jih lahko postavimo v prostor. To nam prihrani precej dela. Dovoljuje nam tudi, da hitro oblikujemo uporabniški vmesnik, poleg tega pa si lahko iz njihove trgovine brezplačno prenesemo paket Standard Assets, ki vsebuje nekaj osnovnih gradnikov in skriptov. Paket Standard Assets nam tako še dodatno olajša delo — pri izdelavi aplikacije smo iz tega paketa uporabili skript za prvoosebni nadzor.

3.1.2 Mono C#

C# je programski jezik, ki ga je razvilo podjetje Microsoft[26]. Gre za objektno usmerjen programski jezik iz družine C-jevskih jezikov. C# samodejno

upravlja s pomnilnikom in je preprost za uporabo. C# je sicer del Microsoftovega ogrodja .NET, vendar pa se je v letu 2001 pojavila odprtokodna alternativa za .NET, imenovana Mono[3]. Medtem ko je ogrodje .NET razvito za operacijske sisteme Windows, lahko Mono aplikacije tečejo tudi na operacijskih sistemih Linux in MacOS. Ogrodje Mono sicer ne podpira nekaterih funkcionalnosti ogrodja .NET[4], vendar pa nam to ni predstavljalo nobenih težav.

C# smo uporabili za programiranje glavne aplikacije.

3.1.3 Python 2

Python[9] je programski jezik, ki ga razvija Python Software Foundation. Je enostaven za uporabo in omogoča hitro izdelavo programov. Python je razširljiv z uporabo raznih knjižnic, ki jih lahko na enostaven način namestimo in uvozimo v naš program. Uporabili smo ga predvsem pri izdelavi pomožnih programov.

3.1.4 FFmpeg

FFmpeg[1] je prosto dostopno in vsestransko multimedijsko ogrodje, s katerim lahko kodiramo, odkodiramo, prekodiramo, filtriramo in predvajamo veliko različnih medijskih formatov, tako glasbe kot tudi videa. FFmpeg je na voljo na vseh večjih računalniških operacijskih sistemih — tako na Linuxu, kot tudi na operacijskih sistemih Windows in MacOS.

FFmpeg smo uporabili za pretakanje skladb iz storitve Glasba Google Play (GPM) in njihov zapis v format .ogg ter za pretvarjanje skladb, prenesenih od ponudnika 7digital, v format .wav.

Poglavje 4

Izvedba rešitve

V tem poglavju je podrobneje opisan razvoj naše aplikacije. Izvedba je imela v grobem tri glavne dele: računanje podobnosti med skladbami, izdelavo 3D prostora ter prenašanje, pretakanje in predvajanje skladb v programu. Ker programski jeziki in knjižnice, ki smo jih uporabljali, za poimenovanje spremenljivk in funkcij uporabljajo angleške izraze, smo zaradi doslednosti v kodi uporabljali angleščino. Uporabniški vmesnik aplikacije je prav tako v angleščini, saj smo želeli aplikacijo javno objaviti.

4.1 Cilji

Tekom diplomskega dela smo želeli izdelali aplikacijo, ki preslika glasbeno zbirko v prostor glede na podobnosti med skladbami, uporabniku pa dovoljuje, da glasbeno zbirko raziskuje s premikanjem po pokrajini. Pri tem smo si zadali naslednje cilje:

3D pokrajina – želeli smo, da lahko glasbeno zbirko uporabnik raziskuje v 3D prostoru, v katerem so skladbe razvrščene glede na medsebojno podobnost.

Velikost zbirke – aplikacija naj bo zmožna predstavljati večmiljonske zbirke skladb.

Poslušanje skladb – medtem ko se uporabnik sprehaja skozi pokrajino, naj se predvajajo skladbe, ki predstavljajo uporabnikovo okolico.

Možnost podrobnega raziskovanja – če so uporabniku všeč skladbe na nekem območju, mu ponudimo možnost, da svojo bližnjo okolico razišče bolj podrobno.

Podpora več operacijskim sistemom – aplikacija naj ne bo vezana na samo en operacijski sistem.

4.2 Podatki

Ker smo želeli imeti v našem programu karseda veliko skladb, računanje značilk samih pa ni bil eden izmed ciljev našega dela, smo za podatke zaprosili univerzo Johannesa Keplerja v Linzu. Ti so nam posredovali štiri milijone značilk skladb, vendar pa nam je zaradi poškodovanega arhiva uspelo pridobiti le tri milijone značilk. Skupaj z značilkami smo dobili tudi metapodatke o skladbah, ki so vključevali izvajalca, naslov skladbe ter identifikacijsko številko skladbe pri ponudniku 7digital. Poleg podatkov so nam posredovali tudi kodo za dekodiranje značilk v programskem jeziku Python.

Eden izmed ciljev je bil tudi, da bi se med sprehajanjem po prostoru predvajale skladbe, ki so nam blizu. Ker bi bila prenos in shranjevanje tako velikega števila skladb zahtevna, smo se odločili skladbe sproti prenašati iz interneta. Sprva smo nameravali uporabljati ponudnika storitev 7digital, saj smo za vsako skladbo imeli identifikacijsko številko skladbe. Ta pristop bi bil boljši, saj identifikacijska številka enolično določa skladbo. Ponudnik prav tako ponuja zastonj, a omejen dostop do njihovega APIja. Za dostop smo se prijavili na njihovi spletni strani, vendar pa se je API izkazal za omejenega. Na njem smo bili omejeni na nekatere regije, iz katerih nismo mogli dostopati do vseh skladb. Prav tako smo bili omejeni na 4000 zahtevkov na dan. Ker o dostopnosti skladbe nismo vedeli ničesar, dokler do nje nismo mogli dostopati, se je zaradi velikega števila neuspelih zahtevkov omejitev izkazala

za precej nizko.

Zaradi tega smo kasneje začeli uporabljati tudi storitev Glasba Google Play (GPM), ki je sicer plačljiva, vendar pa nam je bilo na njej na voljo precej več skladb. Ker za storitev GPM nismo imeli podanega identifikatorja, smo na njej skladbo iskali s kombinacijo imena in izvajalca. Težava tega pristopa je ta, da kombinacija naslova skladbe in izvajalca ni enolična, saj si lahko več izvedb iste skladbe deli oboje. Primerov tega je ogromno: "The Sunk'n Norwegian" škotskega izvajalca Alestorm ima na storitvi GPM dve različici z enakim naslovom — eno v albumu "Back Through Time", drugo pa v albumu "Live At The End of the World" (iskanje najde še tretjo in četrto različico, vendar pa imata ti na storitvi GPM spremenjena naslova). Še hujši primer je pesem "Kingston Vampires" izvajalcev DJ Fresh in Pendulum, ki ima dve različici v istem albumu — edina razlika med njima pa je njuna dolžina.

Na srečo v večini primerih pesem z istim naslovom v različnih albumih zveni povsem enako (vsaj za človeškega poslušalca), remiksi in akustične ali instrumentalne izvedbe pa imajo na spletnih storitvah za poslušanje glasbe praviloma v naslovu zavedeno, da gre za remiks ali drugačno izvedbo. Zaradi tega smo se odločili, da je ta pristop kljub nepopolnosti dovolj natančen, da ga uporabimo.

4.3 Računanje podobnosti

Podobnosti smo računali z metodo t-SNE. Za t-SNE smo se odločili, ker najbolje ohranja strukturo podatkov. Ker je implementacij t-SNE več, smo se odločili za Barnes-Hut implementacijo (BH t-SNE)[24]. Ta je trenutno najhitrejša implementacija t-SNE, ki je tudi bolj primerna za velike zbirke podatkov.

Program za računanje BH t-SNE je bil spisan v programskem jeziku C++, vendar pa smo z njim upravljali preko programa, napisanega v Pythonu. Ta je skrbel za branje vhodnih podatkov in pakiranje v obliko, primerno za glavni program. Prav tako je program z metodo PCA zmanjšal število

začetnih dimenzij iz 980 na 30.

Težava Python programa je v tem, da vse podatke najprej prebere v delovni pomnilnik, zaradi česar posledično porabi veliko pomnilniškega prostora. Za procesiranje podobnosti med dvesto tisoč skladbami porabi več kot 16 GB glavnega pomnilnika, kar je občutno preveč — še posebno glede na to, da ga ima povprečen računalnik na voljo precej manj. Seveda bi lahko programa spremenili tako, da bi uporabljal manj pomnilnika (tako, da bi več podatkov hranili na disku), vendar pa bi to podaljšalo izvajanje algoritma.

To bi bila težava, saj je Barnes-Hut implementacija t-SNE — kljub temu, da je relativno hitra v primerjavi z drugimi implementacijami t-SNE — še vedno prepočasna, da bi podobnosti računali sproti. Še eno težavo je predstavljal prostor, ki so ga zasedale značilke: na disku so vse značilke skupaj zasedale več kot 100 GB prostora.

Zaradi teh težav smo se odločili, da bomo podobnosti med skladbami izračunali vnaprej na strežniku, ki ima več pomnilnika in izmenljivega prostora (swap), vendar pa smo se tudi na strežniku zaradi omejenega prostora omejili le na malo več kot milijon skladb. Seznam skladb, za katere smo računali podobnosti, je vseboval prvih milijon skladb, ki so bile navedene v datoteki z metapodatki. Temu smo dodali še 22295 skladb, ki smo jih izbrali v skladu z našimi osebnimi željami.

Rezultat računanja podobnosti so bile koordinate v 2D prostoru za vsako skladbo.

4.4 Priprava prostora

Če želimo na podlagi dobljenih točk ustvariti pokrajino, ne zadošča le umeščanje dobljenih točk v prostor. Čeprav je točk skupaj veliko, jih je v posamezni enoti prostora bolj malo. Če vsako skladbo ponazorimo s kocko, so stebri kock nezvezni, med njimi pa je polno lukenj. Če bi pokrajino ustvarili tako, da bi višino tal na določenem mestu določili z višino stebra kock, potem bi bila tako dobljena pokrajina šumnata in nazobčana zmešnjava, ki uporabniku

ne bi povedala veliko. Zaradi tega je bilo potrebno dobljene točke najprej nekako obdelati.

Točkam smo zato takoj po računanju dodali identifikacijsko številko ponudnika 7digital, ki je določala, katero skladbo ta točka predstavlja. Ob prvem zagonu programa smo jih prebrali iz tekstovne datoteke ter jih za nadaljnjo uporabo prepisali v binarno datoteko. V nadaljnjih zagonih smo točke brali iz binarne datoteke, saj je bilo tako branje nekoliko hitrejše.

Točke smo v programu hranili kot objekt, v katerem smo hranili koordinate točke in njeno identifikacijsko številko. Kasneje smo v tem objektu hranili še metapodatke o skladbi, ki smo jih prebrali naknadno. Metapodatki so vključevali naslov skladbe ter ime izvajalca — torej podatke, ki smo jih dobili skupaj z značilkami. Dodatnih podatkov nismo dodajali.

Metapodatki so vsebovali samo znake iz nabora ASCII. Znaki, ki niso del nabora ASCII, so bili predstavljeni z ubežnimi kodami HTML. Čeprav C# vsebuje razred za dekodiranje teh kod, nam v Unityju ni dostopen — vendar pa je uporabnik Cratesmith na spletni strani objavil različico razredov, prilagojen za uporabo z Unity[12]. Razrede za dekodiranje kod smo tako prenesli iz njegovega repozitorija.

4.4.1 Izgradnja pokrajine in umeščanje skladb v prostor

Ena izmed bolj enostavnih idej pri naključnem ustvarjanju pokrajin je ta, da se generira več šumnatih slik v različnih velikostih. Manjše slike se nato poveča na velikost največje, vsaki sliki se doda neko utež, na koncu pa se slike sešteje in povpreči. Slike manjših velikosti nam v tem primeru priskrbijo grobo obliko pokrajine, večje slike pa na grobo obliko pokrajine dodajo podrobnosti.

Podobno strategijo lahko uberemo tudi tu, le da namesto naključnih števil preštejemo število točk v določenem območju. To najbolj enostavno dosežemo z uporabo štiriškega drevesa (quad tree). Točke umestimo v drevo, pri čemer na vsakem nivoju štejemo število skladb v podnivojih.

V ta namen smo implementirali preprosto obliko štiriškega drevesa. Drevo je imelo vnaprej določeno število nivojev. List je predstavljal prostor velikosti 1 x 1 enote, velikost območja, ki ga pokriva drevo, pa je bila določena s številom nivojev. Drevo je sprejemalo točke s koordinatama med 0 in 2^n (kjer je n število nivojev), uvrščanje točk v poddrevesa pa je bilo odvisno od tega, ali sta bili koordinati večji ali manjši od sredinske točke. Točke smo hranili le v listih drevesa, vendar pa smo v implementaciji drevesa vključili tudi funkcije, ki so zgradile seznam točk za poljubno kvadratno območje. Implementacija je vsebovala tudi funkcijo, ki je lahko za vsak nivo vrnila nenormaliziran višinski zemljevid.

Ker je bilo drevo vselej fiksne velikosti, smo pred vstavljanjem točk v drevo koordinate točk normalizirali na območje, ki ga je pokrivalo drevo. To nam je kasneje omogočilo enostaven prehod med stopnjami podobnosti.

Ko smo v drevo vključili vse točke, smo iz njega po prej opisanem postopku ustvarili teren. Izbrali smo si štiri nivoje, na katerih smo zgradili višinski zemljevid. Vsakega smo normalizirali in ga z uporabo Lanczosovega filtriranja povečali na velikost terena. Za spreminjanje velikosti smo uporabili knjižnico [27]. Vsakemu nivoju smo določili nek faktor, s katerim smo določili moč vpliva nivoja na videz pokrajine, ter jih sešteli. Zatem smo še zadnjič normalizirali vrednosti. Pri normalizaciji smo najprej podali najvišjo želeno višino (ki je bila manjša od najvišje višine, ki bi jo lahko na terenu prikazali), nato pa višine normalizirali tako, da je bila večina pokrajine pod to višino. Pri tem smo dovolili, da je bilo 0,25 % pokrajine nad najvišjo želeno višino, saj so se v praksi pojavile anomalije. Izkazalo se je namreč, da je bilo med uporabljenimi skladbami tudi več izvedenk pesmi 'Vse najboljše,' ki so se razlikovale le po imenu slavljenca. Zaradi velike medsebojne podobnosti so bile na višjih nivojih vse točke, ki so predstavljale različice te pesmi, na istem mestu; posledično pa so ustvarile nenormalno visok vrh. Če bi vsilili najvišjo višino brez upoštevanja anomalij, potem bi drugi vrhovi prišli do izraza precej manj, kot bi si želeli.

4.4.2 Podrobnejše raziskovanje prostora

Zaradi velikega števila skladb, predstavljenega s pokrajino, je pokrajina na najvišjem nivoju dobro predstavljala razliko med različnimi žanri, medtem ko razlika znotraj posameznih žanrov in gruč ni prišla do izraza. Zato smo potrebovali način, kako podrobneje predstaviti razlike znotraj nekega območja.

Za bolj enostavno reševanje tega problema smo uvedli večje kose (chunks), ki so predstavljali območje velikosti 32 x 32 enot — na petem nivoju drevesa. Ko uporabnik želi podrobneje raziskovati posamezno območje, iz drevesa vzamemo vse točke, ki pripadajo kosu, v katerem se nahaja uporabnik, in njegovim sosedom. Te točke dodamo v seznam ter iz njega zgradimo pokrajino na enak način, kot smo to storili za prvi nivo.

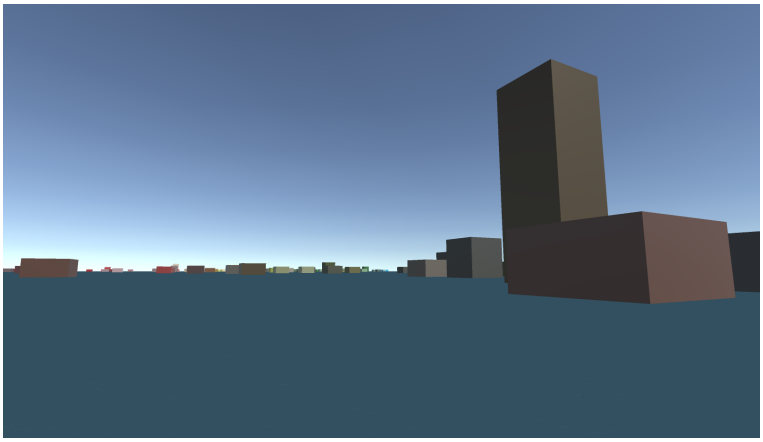
Pravila za izgradnjo pokrajine se spreminjajo glede na število točk v območju. Če je točk manj kot 15.000, se spremenijo nivoji drevesa in pripadajoče uteži, ki jih uporabimo za izgradnjo pokrajine. Najnižji nivo pokrajine dosežemo, če je bilo v kosih okoli igralca manj kot 4000 točk. V tem primeru tudi ne zgradimo pokrajine, temveč pesmi predstavimo s kockami. Kocka tu ne predstavlja posamezne pesmi, temveč manjše področje. Če se zgodi, da je na področju kocke več kot ena pesem, potem jo razpotegnemo v kvader, višina katerega je sorazmerna s številom pesmi na območju.

Za lažjo vrnitev v višje nivoje hranimo točke iz vseh predhodnih nivojev.

4.4.3 Videz pokrajine

Privzeto je bil teren bele barve, saj mu nismo dodali nobene teksture. Kot tak ni bil le dolgočasen, temveč je bilo oteženo tudi orientiranje v prostoru. Podobne primere bi lahko imeli, če bi preprosto izbrali nekaj tekstur, ki bi jih mešali med sabo, zato smo se odločili, da bomo teksturo naredili sami.

Za namene izdelave teksture smo najprej poiskali vrhove. To smo storili tako, da smo vzeli višinski zemljevid na eni izmed bolj grobih stopenj ter na njem poiskali lokalne maksimume. Mesta lokalnih maksimumov smo zatem razvrstili od najvišjega do najmanjšega in zavrgli tiste, ki so ležali nižje od



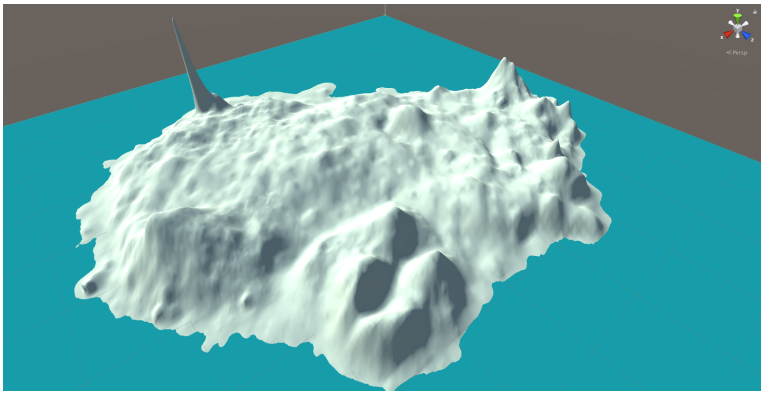
Slika 4.1: Predstavitev pesmi s kockami. Če se na istem mestu nahaja več kot ena pesem, potem kocko razpotegnemo v kvader.

določene višine (tako smo se znebili nizkih vrhov, ki niso vsebovali omembe vrednega števila točk). Zatem smo ustvarili seznam, na katerega smo dodajali dobljene vrhe od najvišjega do najnižjega. Pri tem smo preverjali, če so vrhovi dovolj oddaljeni eden od drugega. Če je bil nek vrh preblizu kateremu izmed višjih vrhov, ki smo jih prej dodali na seznam, smo ga zavrgli.

Vsakemu vrhu smo tudi dodelili barvo. Barva vrha je bila večinoma določena z njegovim mestom: ena izmed koordinat je določala obarvanost, druga pa svetlost. Določanje barve je imelo tudi manjšo naključno komponento, saj smo vrednost in obarvanost v manjši meri spreminjali tudi z naključnimi števili. Nasičenost je bila določena s kombinacijo naključnosti in svetlosti: temne barve so bile manj nasičene.

Če je bil vrh dovolj visok, smo ga pokrili s snegom. Tudi barva snega je bila določena s kombinacijo naključnosti in pravil: svetlost je bila naključna (med 75-100 %), obarvanost snega se je razlikovala od obarvanosti vrha za neko manjše naključno število. Nasičenost je bila tudi tukaj pogojena s svetlostjo: višja svetlost je tu pomenila manj nasičenosti, razen pri modrih odtenkih.

Seznam vrhov smo nato poslali v funkciji, ki je ustvarila teksturo na

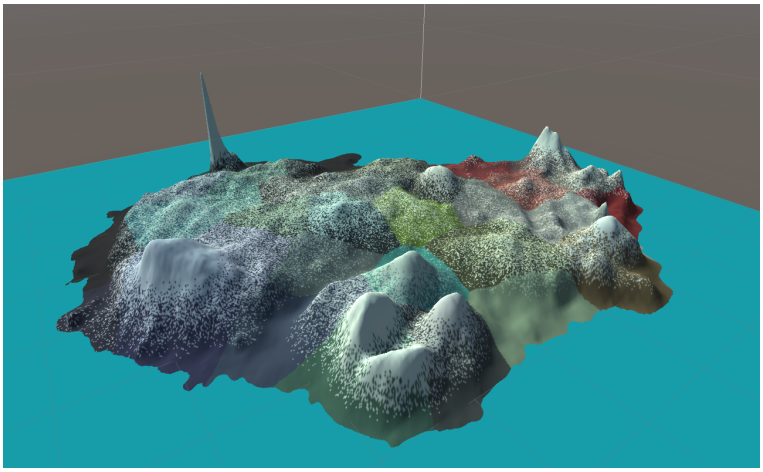


Slika 4.2: Teren brez tekstur je videti monoton.

podlagi Voronojevih diagramov. Za računanje Voronojevih diagramov smo uporabili kar naivni način, ki smo ga malo aproksimirali. Pokrajino smo zopet razdelili na kose, ki so bili manjši od najmanjše dovoljene razdalje med vrhovi. Za vsak kos smo izračunali tri vrhove, ki so ležali najbližje njegovi sredini. Potem smo za vsako točko v kosu preverili, kateremu izmed teh treh vrhov je najbližja in jo pobarvali z barvo pripadajočega vrha. V tej funkciji smo tudi določili, ali je vrh zasnežen ali ne. Definirali smo dve ravnini. Če je točka ležala pod nižjo ravnino, je bila tista točka obarvana z osnovno barvo vrha. Če je bila točka nad višjo ravnino, potem je bila tekstura obarvana z barvo snega, ki je bila značilna za tisti vrh. Med ravninama je bila zasneženost določena naključno, verjetnost, da je bila točka pokrita s snegom, pa je bila odvisna od višine terena na tistem mestu.

4.5 Predvajanje skladb

Eden izmed ciljev je bil tudi ta, da uporabnik med sprehodom po pokrajini posluša skladbe, ki se nahajajo v njegovi bližini. Ker skladb zaradi njihovega velikega števila ni bilo smiselno shranjevati na disku, smo se odločili, da jih bomo po potrebi prenašali prek interneta. To nas omejuje v kar nekaj pogledih, saj lahko prekomerno prenašanje skladb iz interneta krši pogoje



Slika 4.3: Pokrajina, pobarvana z različnimi barvami. Tako smo rešili težavo preveč monotonega videza.

ponudnika storitev. Treba se je tudi zavedati, da so hitrosti interneta lahko pri določenih uporabnikih zelo počasne, zato se je pri prenašanju podatkov pametno omejiti. Z namenom zmanjšanja števila prenosov smo se odločili, da bomo hkrati predvajali kvečjemu eno skladbo. To je pomenilo, da se je bilo potrebno odločiti, katere skladbe se bodo predvajale ob katerem trenutku.

4.5.1 Iskanje najbolj primerne skladbe

Iskanje skladbe bi lahko izvedli na veliko načinov. Lahko bi poskusili določiti gruče in na podlagi koordinat točk, na katerih se skladbe nahajajo, določiti "povprečno" skladbo. Problem tega pristopa je, da meje med gručami niso povsem jasne, zato je težko reči, kje se konča ena gruča in začne druga. Lahko bi določili vrhove ter predvajali tisto skladbo, ki se nahaja najbližje nam najbližjemu vrhu. S pomočjo vrhov bi lahko definirali Voronojev diagram ter v vsakem poligonu določili povprečno skladbo ter jo predvajali.

Odločili smo se za še bolj enostavno možnost. Vzeli smo kose, na podlagi katerih smo ustvarili bolj podrobne nivoje pokrajine, ter izračunali povprečje koordinat vseh točk. Skladbe smo zatem razvrstili po oddaljenosti do pov-

prečne koordinate. Najbližja skladba je bila uporabljena kot tista, ki jo bo uporabnik slišal ob vstopu v ta kos pokrajine. V kolikor ta skladba ne bi bila na voljo za prenos, bi uporabili drugo najbližjo.

4.5.2 Prenašanje skladb iz interneta

Skladbe smo od ponudnikov prenašali z uporabo programov, napisanih v jeziku Python — sprva od ponudnika 7digital, kasneje pa iz storitve Glasba Google Play (GPM). Pri obeh ponudnikih so skladbe na voljo v formatu mp3. Za prenašanje skladb od ponudnika 7digital smo uporabili program download7digital.py[10], ki smo ga priredili svojim zahtevam. Ta je prenos datotek upravljal namesto nas, zato smo skladbe bližnjih kosov prenesli že vnaprej. Med sprehajanjem po pokrajini bi naš vmesnik v ozadju prenašal skladbe sosednjih kosov pokrajine z namenom, da bi nam bile skladbe na voljo takoj, ko bi zapustili trenuten kos. Ko se je izkazalo, da je API ponudnika 7digital precej omejen, smo ta način opustili in poskusili skladbe prenašati iz storitve GPM. Za prenos skladb iz storitve GPM smo uporabili knjižnico gmusicapi.

Knjižnica gmusicapi[2] je implementacija neuradnega vmesnika API za GPM. Ponuja nam tri vmesnike: vmesnik za spletni klient (Webclient), vmesnik za mobilni klient (Mobileclient) in vmesnik za Music Manager[1] (Musicmanager). Od teh treh možnosti vmesnika Mobileclient in Webclient ponujata funkcionalnosti, ki jih potrebujemo. Ker je vmesnik Webclient manj razvit in slabše testiran, smo se odločili za uporabo vmesnika Mobileclient.

Preden lahko uporabimo storitev GPM, se moramo vanjo najprej prijaviti. Ukaz za prijavo sprejema najmanj tri argumente: uporabniško ime, geslo ter identifikacijsko številko naprave. Če želimo iz storitve prenašati skladbe, moramo za identifikacijsko številko uporabiti identifikacijsko številko ene izmed naših mobilnih (Android ali iOS) naprav. Ker ob prvi prijavi te številke ne vemo, lahko namesto nje uporabimo MAC naslov naprave. Naprave, ki so povezane z našim računom, lahko dobimo s klicem metode

[1]Music Manager je program, s katerim lahko na storitev GPM nalagamo svojo glasbo

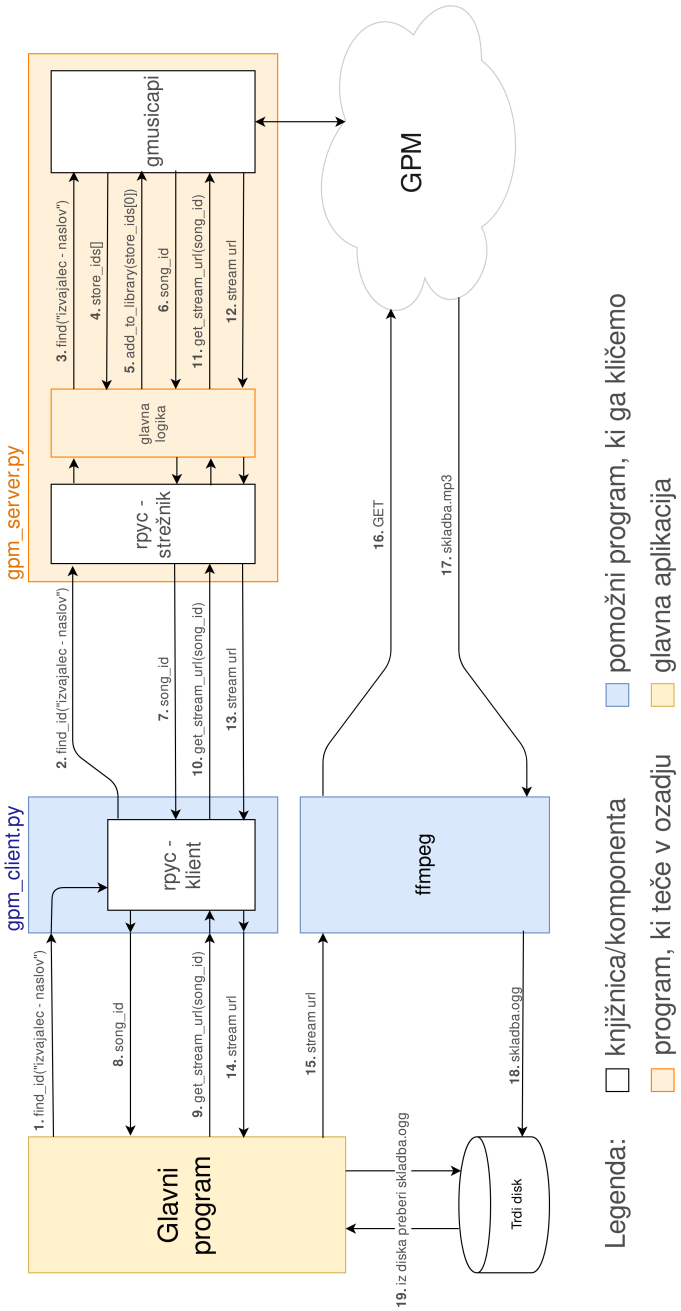
`Mobileclient.get_registered_devices()`. Na seznamu naprav, ki jih vrne ta metoda, sta navedena tudi tip naprave ter njena identifikacijska številka. Za nadaljnje prijave si izberemo številko ene izmed naprav tipa `ANDROID` oziroma `iOS`.

Ko se v storitev GPM prijavimo z uporabo identifikacijske številke veljavne naprave in če smo na storitev naročeni, potem lahko iščemo po glasbenem katalogu storitve GPM.

Mobileclient nam ponuja funkcijo `search`, z uporabo katere lahko v storitvi GPM iščemo pesmi, izvajalce, albume in tako naprej. Ta funkcija nam vrne objekt, ki vsebuje nekaj vrst zadetkov (npr. `artist_hits` vključuje vse izvajalce, ki se ujemajo z iskanim izrazom, `album_hits` vključuje albume itd.). Ker nas zanimajo le posamezne pesmi, so za nas pomembni le zadetki, ki so v polju `song_hits`. Če je polje `song_hits` prazno, potem iskane pesmi na storitvi GPM ni. Če polje `song_hits` ni prazno, pa vselej uporabimo prvo skladbo v njem.

Vsaka skladba ima nekaj identifikacijskih številk, ki služijo različnim namenom. Z identifikacijskimi številkami skladb, ki jih vrne funkcija `search`, ne moremo zahtevati naslova, s katerega lahko prenesemo skladbo. Tako številko imajo le skladbe, ki smo jih dodali v svojo knjižnico, zato moramo zadetke iskanja pred predvajanjem dodati v svojo knjižnico. Šele ko je skladba dodana v knjižnico, imamo na voljo identifikacijsko številko skladbe, s katero lahko zahtevamo naslov, iz katerega jo lahko prenesemo. Ob uspešnem iskanju zato prvo skladbo na seznamu `song_hits` dodamo v knjižnico, identifikacijsko številko skladbe pa pošljemo glavnemu programu.

Ukaz za pridobitev naslova, iz katerega lahko prenesemo skladbo, moramo izdati posebej. Tako smo se odločili zato, da bi lahko skladbe iskali že vnaprej in neodvisno od prenašanja. Za pridobitev tega naslova pa potrebujemo identifikacijsko številko, ki smo jo dobili v prejšnjem koraku. Ko enkrat dobimo naslov, moramo prenos skladbe izvesti sami. Za prenos skladb zaradi težav, opisanih v naslednjem odseku, uporabljamo multimedijsko ogrodje FFmpeg.



Slika 4.4: Diagram, ki orisuje potek prenosa skladb od ponudnika GPM.

Ker smo v storitev GPM prijavljeni le tekom izvajanja programa in ker nočemo za vsak ukaz posebej izvesti nove prijave, mora naš program za komunikacijo s storitvijo GPM ves čas teči v ozadju. Najlažji način, da programu v programskem jeziku python med izvajanjem posredujemo ukaze, je uporaba knjižnice RPyC[7]. Ta knjižnica nam dovoljuje, da določene funkcije izpostavimo navzven, da so dosegljive drugim programom, ki uporabljajo knjižnico RPyC. Program, s katerim smo upravljali s storitvijo GPM, smo zato napisali tako, da se je obnašal kot strežnik. Zatem smo napisali še drug program, klient, ki je deloval kot vmesnik med glavnim programom ter nadzornim programom za GPM. Namen klienta je bil, da pošlje parametre nadzornemu programu, počaka na odgovor, vrne odgovor glavnemu programu in se nato zaključi.

Ker moramo vsako skladbo pred predvajanjem dodati v knjižnico, naš program v knjižnico dodaja neželene programe. Ker uporabnik večine skladb, predvajanih tekom uporabe vmesnika, verjetno ne bo želel v svoji glasbeni knjižnici, naš nadzorni program vodi seznam skladb, ki smo jih med uporabo programa na novo dodali v uporabnikovo knjižnico. Po končani uporabi lahko nadzornemu programu z uporabo klienta pošljemo ukaz, da iz knjižnice odstrani vse novo dodane skladbe.

Postopek pridobivanja skladb od ponudnika GPM je opisan na diagramu na sliki 4.4.

4.5.3 Predvajanje skladb v Unity

Čeprav Unity podpira veliko glasbenih formatov, predvajanje glasbe v večini teh formatov ni podprto med izvajanjem samega programa. Nalaganje skladb v formatu .mp3 — ki ga uporabljata tako 7digital in GPM — je med izvajanjem programa podprto le na mobilnih platformah. Če želimo to funkcionalnost tudi na računalniku, bi za dekodiranje formata morali napisati svojo knjižnico ali pa uporabiti eno izmed prosto dostopnih. Težava je v tem, da te knjižnice pogosto temeljijo na tehnologiji .NET, uporabi katere bi se rajši izognili. Tako smo uporabili tretjo možnost: z uporabo ogrodja FFmpeg smo

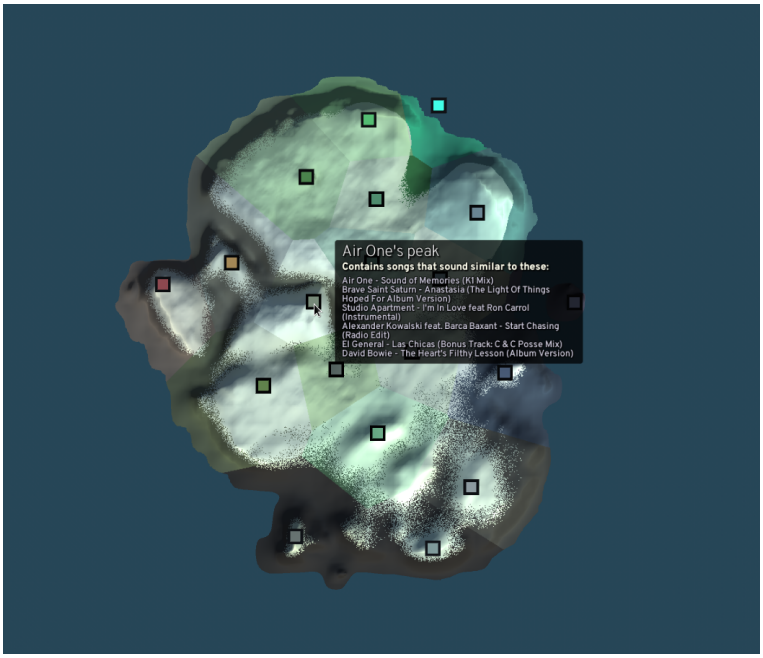
skladbe pretvorili v formata, ki ju lahko beremo tudi med izvajanjem programa: .wav in .ogg. Izbira formata je bila odvisna od izbranega ponudnika storitev.

Format .wav je večji in nekompresiran, vendar pa je pretvorba v ta format opravljena hitreje. Ta format smo uporabili pri skladbah, prenesenih od ponudnika 7digital, saj so bile datoteke, ki jih je bilo potrebno pretvoriti, že na disku.

Format .ogg je kompresiran in zasede manj prostora, vendar pa potrebujemo več časa za pretvarjanje v ta format. Na našo srečo FFmpeg podpira pretakanje medijskih vsebin, kar nam dovoljuje, da pretvorbo v format .ogg opravimo že med prenosom iz interneta. V tem primeru je čas, potreben za pretvorbo, zanemarljiv. Ker nam je storitev GPM ponudila povezavo, prek katere smo lahko prenesli povezavo, smo pri prenosih iz GPM uporabili ta pristop.

Za razliko od prenašanja skladb od ponudnika 7digital, kjer je uporabljen Python program ustvaril datoteko šele po končanem prenosu, je v pristopu, ki smo ga uporabili s storitvijo GPM, del datoteke na voljo na našem disku še preden je prenos končan. Zaradi tega lahko datoteko začnemo predvajati, še preden je prenesena v celoti. Posledica te lastnosti je to, da lahko prenos skladbe zahtevamo šele takrat, ko stopimo v drug kos pokrajine.

Ko želimo v vmesniku, zgrajenem z Unity, predvajati pesem, moramo najprej spraviti zvočno datoteko v objekt tipa AudioClip. To s sabo prinese še eno težavo, ki ima sicer preprosto rešitev. Če želimo predvajati glasbeno datoteko, ki še ni bila prenesena v celoti, bo Unity predvajal le tisti del zvočne datoteke, ki je bil na disku ob trenutku, ko smo ustvarili AudioClip. To težavo smo rešili tako, da smo na vsake toliko časa ponovno ustvarili AudioClip z datoteko, ki smo jo predvajali v tistem trenutku. Zatem smo primerjali število vzorcev v AudioClipu, ki smo ga predvajali s številom vzorcev v novem. Če sta se števili vzorcev razlikovali, smo zamenjali trenutno predvajani AudioClip z novim, predvajanje pa nadaljevali tam, kjer smo končali.



Slika 4.5: Tako je videti pogled zemljevida.

4.6 Uporabniški vmesnik

4.6.1 Raziskovanje pokrajine

Uporabnik raziskuje teren v prvoosebnem pogledu. Za prvoosebni pogled smo uporabili skript 'Rigidbody First Person Controller' iz paketa 'Standard Assets,' ki ga je moč zastonj prenesti iz trgovine Unity. Za gibanje lahko uporabnik uporablja tipke ESDF (E in D za naprej in nazaj, S in F za levo in desno), preslednico (skok) in dvigalko (spremeni hitrost premikanja). Do nižjih nivojev lahko dostopa s tipko W, nazaj na višji nivo pa se vrača s tipko A. Zemljevid je prikazan na sliki 4.5.

Ker iz pokrajine ni mogoče razpoznati, kakšne vrste skladba se nahaja na katerem območju, smo v vmesnik dodali tudi zemljevid, do katerega se lahko dostopa s tipko M. Vrhovi so na zemljevidu predstavljeni z obarvanimi kockami, ki uporabniku kot take ne povedo nič; a ko se uporabnik s kazalcem miške dovolj približa kateri izmed kock, se zraven nje izpiše do devet naslovov

Slika 4.6: Nad vrhovi se v prvoosebnem pogledu prikazujejo njihova imena. Najvišja točka vrha je označena z drevesom. Na teksturi so vidne tudi meje med različnimi kosi.

skladb, ki jih je mogoče najti na tistem območju. Ker se uporabnik lahko med točkami na zemljevidu premika zelo hitro, mu med raziskovanjem zemljevida ne predvajamo pregledov skladb, saj obstaja možnost, da bi med točkami prehajal hitreje, kot bi to omogočala ponudnik storitev ali njegova internetna povezava. Če uporabnik klikne na kocko (ali v njeno neposredno bližino), se premakne na vrh, ki ga kocka predstavlja. V primeru, da uporabnik želi raziskovati svet, brez da bi se prestavil na katerega izmed vrhov, pa lahko zemljevid zapusti s ponovnim pritiskom tipke M ali ubežnice.

Da uporabniku ni treba ves čas odpirati zemljevida, smo nad vrhove tudi v prvoosebnem načinu postavili njihova imena, točko vrha pa smo zaznamovali tudi z drevesom — njegov videz prikazuje slika 4.6. Za še lažje razlikovanje med kosi pokrajine smo na teksturo tal narisali tudi njihove meje.

Na najnižjem nivoju, kjer namesto terena za ponazarjanje porazdelitve skladb uporabljamo kocke, raziskovanje poteka malo drugače. Skladba se tu ne predvaja med sprehajanjem po okolju, temveč se mora uporabnik najprej približati eni izmed kock. Ko je dovolj blizu kocki, lahko s pritiskom na

Slika 4.7: Preprosto okno, ki nam pokaže naslove in izvajalce skladb, ki se nahajajo v določeni kocki.

tipko W odpre okno, v katerem je seznam vseh skladb, ki se v tisti kocki nahajajo. Na temu seznamu lahko uporabnik zahteva, da se začne predvajati ena izmed skladb. Zaradi omejitev implementacije uporabnik ne ve, ali je skladba dejansko na voljo ali ne, dokler ne zahteva njenega predvajanja. Če skladbe ni mogoče dobiti od ponudnika GPM, se ob skladbi namesto belega vprašaja pojavi rdeč križec. Okno s pesmimi je mogoče zapreti s pritiskom ubežnice na tipkovnici. Primer takega okna je prikazan na sliki 4.7.

Slika 4.8: Na zemljevidu lahko hitro vidimo, katera skladba se nahaja v kateri kocki. Če je skladb v kocki več, je v namigu zavedeno število dodatnih skladb.

Poglavje 5

Sklepne ugotovitve

Tekom diplomskega dela smo izdelali aplikacijo, ki preslika glasbeno zbirko v prostor glede na podobnosti med skladbami, uporabniku pa dovoljuje, da glasbeno zbirko raziskuje s premikanjem po pokrajini. Čeprav aplikacija deluje dovolj dobro, obstaja prostor za nadaljnje izboljšave:

Optimizacija izgradnje terena – metoda, ki skrbi za izgradnjo terena, ni najbolje optimizirana. Optimizacije so možne tako pri branju podatkov ob prvem zagonu (branje metapodatkov bi lahko pohitrili, če bi jih zapakirali v binarno obliko), kot tudi pri izgradnji terena. Ker se teren v trenutni izvedbi kljub neoptimalnosti implementacije naloži dovolj hitro, je imela optimizacija nizko prioriteto in je na koncu zaradi pomanjkanja časa nismo izvedli.

Optimizacija pridobivanja skladb – trenutno se iskanje in prenašanje skladbe začne šele, ko uporabnik prestopi mejo med dvema kosoma pokrajine. Zaradi tega se lahko pravilna skladba začne predvajati šele nekaj sekund po tem, ko uporabnik prestopi mejo. To težavo bi lahko rešili tako, da bi zaznali, kdaj se uporabnik približa meji kosa, ter poskušali na storitvi GPM najti — ali pa iz nje celo vnaprej prenesti — skladbo, ki predstavlja sosednji kos.

Drugačen način generiranja nižjih nivojev – nižje nivoje bi lahko zgra-

dili na podlagi območij okoli vrhov, ki smo jih definirali na podlagi Vonorojevih diagramov. S tem bi se rešili kvadratnega videza otoka na nižjih nivojih.

Večja integracija s ponudniki storitev – uporabniku bi lahko dovolili, da skladbe, ki jih tekom predvajanja posluša, ocenjuje ter dodaja na svoje sezname predvajanja. Prav tako bi lahko uporabniku ponudili izbiro med več ponudniki.

Poenostavljenje uporabniške izkušnje – vzpostavitev delovnega okolja za aplikacijo zahteva kar nekaj truda, še posebno na operacijskih sistemih Windows. Potrebno je namreč namestiti Python, vse potrebne Python knjižnice ter na roke zagnati program, ki opravlja komunikacijo s storitvijo GPM. Aplikacija trenutno prav tako ne podpira uporabniku prijaznega načina za prijavo v storitev.

Brskalniška različica aplikacije – trenutna implementacija lahko teče le kot namizna aplikacija, ki jo je pred uporabo potrebno prenesti na računalnik. Iz vidika enostavnosti za uporabo bi bilo lepo, če bi lahko do aplikacije dostopali tudi preko brskalnika. Dostop preko brskalnika bi še dodatno olajšal uporabo aplikacije, saj za uporabo prenos ne bi bil potreben. Zaradi zanašanja na zunanje programe in programe brskalniška različica ni mogoča brez precejšnih sprememb v kodi.

Literatura

[1] Ffmpeg. Dosegljivo: https://www.ffmpeg.org/. [Dostopano: 16. 9. 2017].

[2] gmusicapi: an unofficial api for google play music. Dosegljivo: https://unofficial-google-music-api.readthedocs.io/en/latest/. [Dostopano: 18. 9. 2017].

[3] Mono. Dosegljivo: http://www.mono-project.com/. [Dostopano: 16. 9. 2017].

[4] Mono: Compatibility. Dosegljivo: http://www.mono-project.com/docs/about-mono/compatibility/. [Dostopano: 16. 9. 2017].

[5] The neptune interface. Dosegljivo: http://www.cp.jku.at/projects/neptune/. [Dostopano: 15. 9. 2017].

[6] neptune — slika vmesnika. Dosegljivo: http://www.cp.jku.at/projects/neptune/nepTune_large.jpg. [Dostopano: 21. 9. 2017].

[7] Rpyc - transparent, symmetric distributed computing. Dosegljivo: https://rpyc.readthedocs.io/en/latest/. [Dostopano: 18. 9. 2017].

[8] Unity - game engine. Dosegljivo: https://unity3d.com/. [Dostopano: 18. 9. 2017].

[9] Welcome to python. Dosegljivo: https://www.python.org/. [Dostopano: 16. 9. 2017].

[10] audiolabs. Repozitorij apsrr-2016 uporabnika audiolabs. Dosegljivo: https://github.com/audiolabs/APSRR-2016/tree/master/Bergmann-Raffel. [Dostopano: 18. 9. 2017].

[11] LLC Audiosurf. Audiosurf homepage. Dosegljivo: http://www.audiosurf.com/. [Dostopano: 20. 9. 2017].

[12] Cratesmith. Restsharp-for-unity3d/monohttp. https://github.com/Cratesmith/RestSharp-for-unity3d/tree/master/RestSharp/Extensions/MonoHttp. [Dostopano: 16. 9. 2017].

[13] Markus Frühwirth and Andreas Rauber. *Self-Organizing Maps for Content-Based Music Clustering*, pages 228–233. Springer London, London, 2002.

[14] Dianyuan Han. Comparison of commonly used image interpolation methods. 03 2013.

[15] Peter Knees and Markus Schedl. *Applications*, pages 215–246. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016.

[16] Mojang. Minecraft official site. Dosegljivo: https://minecraft.net/en-us/. [Dostopano: 20. 9. 2017].

[17] Elias Pampalk and Andreas Rauber. Content-based organization and visualization of music archives. Technical report, 2006.

[18] Jim Rossignol. Interview: Codies on fuel. Dosegljivo: https://www.rockpapershotgun.com/2009/02/24/interview-codies-on-fuel/. [Dostopano: 20. 9. 2017].

[19] Klaus Seyerlehner and Markus Schedl. Block-level audio features for music genre classification. 2009.

[20] Klaus Seyerlehner, Markus Schedl, Tim Pohle, and Peter Knees. Using block-level features for genre classification, tag classification and music similarity estimation.

[21] Sebastian Stober and Andreas Nürnberger. *MusicGalaxy: A Multi-focus Zoomable Interface for Multi-facet Exploration of Music Collections*, pages 273–302. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[22] Ken Turkowski. Graphics gems. chapter Filters for Common Resampling Tasks, pages 147–165. Academic Press Professional, Inc., San Diego, CA, USA, 1990.

[23] Beatrix Vad, Daniel Boland, John Williamson, Roderick Murray-Smith, and Peter Berg Steffensen. Design and evaluation of a probabilistic music projection interface, 2015.

[24] Laurens van der Maaten. Barnes-hut t-sne. Dosegljivo: `https://github.com/lvdmaaten/bhtsne`. [Dostopano: 20. 9. 2017].

[25] Laurens van der Maaten and Geoffrey E. Hinton. Visualizing high-dimensional data using t-sne. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

[26] Bill Wagner, Luke Latham, Peter Onderka, and Maria Wenzel. A tour of the c# language. Dosegljivo: `https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/`. [Dostopano: 16. 9. 2017].

[27] Kai Wegner. mono scaling. Dosegljivo: `https://github.com/kwnetzwelt/mono_scaling`. [Dostopano: 16. 9. 2017].